Linux 2

More cool stuff

grep

- This is the coolest command ever. When you start having larger piles of code, grep is your friend.
- The cryptic name stands for "global regular expression print". You use it to scan quickly through any number of files and folders looking for a certain text string. This is really useful when, for example, you change the definition of one of your own functions and need to find everywhere you have used it.
- More info: https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux#literal

grep, cont.

- This is all done from the linux command line.
- The general usage is:
- grep [string you are looking for] [file(s) to look in]
- Here is an example of looking for the string "interp_" in all the python code in a folder:

```
Parkers-MacBook-Pro:tests_examples pm7$ grep -n interp_ *.py
test_2d_interp.py:4:Specifically I am benchmarking my own zfun.interp_scattered on plaid()
test_2d_interp.py:66:    # note: zfun.interp_scattered_on_plaid() uses zfun.get_interpolant()
test_2d_interp.py:67:    zs = zfun.interp_scattered_on_plaid(xs, ys, xvec, yvec, Z)
test_2d_interp_cas6.py:4:Specifically I am benchmarking my own zfun.interp_scattered on plaid()
test_2d_interp_cas6.py:64:    # note: zfun.interp_scattered_on_plaid() uses zfun.get_interpolant()
test_2d_interp_cas6.py:65:    zs = zfun.interp_scattered_on_plaid(xs, ys, xvec, yvec, Z)
test_3d_interp_cas6_make_tree.py:6:test_3d_interp_cas6.py.
```

grep

- The "-n" flag means it tells you the line numbers in the files.
- If you want to search recursively down directory trees use "-r".
- I also sometimes find it useful to skip large binary files (like NetCDF model output). For this use "-I".
- Hence my typical use is: grep –rnI ...
- The * in "*.py" on the last slide is an example of a "wildcard" and it means "anything". So in this case ALL files whose filename ends in ".py" will be searched by grep. This is an example of how you make your life easier by having consistent suffixes, even though you don't need them.

Wildcards, Regular Expressions

- Here is where the power of linux commands really starts to shine. Regular expressions are just rules for using a very flexible set of wildcards to refer to subsets of things. They can be used with folder names, filenames, and strings inside of files. Hence you can use them both in the [string] and in the [file(s)] you grep.
- You can also use regular expressions with other linux commands, like ls or rm.
- Examples:
- *.nc = all filenames ending in .nc
- foo_[0-9].nc = all files in the sequence foo_1.nc to foo_9.nc
- foo_[0-9][0-9].nc = all files in the sequence foo_00.nc to foo_99.nc
- You can also search for letter ranges with [a-z] or [A-Z]
- Use quotes around a string when searching for strings that have things like quotes and square brackets, BUT in this case you have to use the "F" flag, for example:

```
Parkers-MacBook-Pro:tests_examples pm7$ grep -nF "smooth[" *.py
test_godin.py:33:smooth[:n] = np.nan
test_godin.py:34:smooth[-n:] = np.nan
```

- There is much more to learn about regular expressions, but you will have to pick it up along the way.

pipe |

- Linux commands can be strung together with the "pipe" symbol: |
- The place I use this **all the time** is to look through my history to find specific commands I issued in the recent past:
- `history | grep [string to search for]`
- Try this out yourself! It basically takes the output of history and routes it through a grep search. So cool.

Scripting

- A shell script is just a file that contains a sequence of linux commands, like any program. It also has basic programming structures like loops and if statements.
- Writing shell scripts is a large subject, too large for me to really cover thoroughly. In addition I am not enough of an expert.
- You RARELY need to write shell scripts, but when you do they can save you a lot of time, even though you might have to spend effort on each line (and searching Stack Overflow).
- Here is an easy to use set of examples for scripting:
- https://devhints.io/bash
- Here is a free source of authoritative info, although it is difficult to read:
- https://www.gnu.org/software/bash/manual/

Examples of what you can do with a shell script

- Go through a large number of files line by line and extract specific information
- Make lots of sub-directories and move files into them in organized ways
- Make a driver to run a sequence of jobs. These can be python programs, running a model, moving input and output.
- Exercise: take of some part of your research and break it into a sequence of tasks. Write them down. In general you should always start a coding job by writing what you want to do. Don't just start typing code!!!

Exercise

- Find the file first_script.sh on the web site and put it in its own folder.
- In general I do not put my working code in a mirrored place like Dropbox. Instead I use GitHub for backing up code and managing it across machines (much more about GitHub later!).
- Make the directory name something simple and Linux-y. I called mine "pmec" standing for Parker MacCready Effective Computing.
- Read through the code in the shell script.
- Then try running it from the command line. Do the results make sense? Did it create another directory?